APPLYING STATE-OF-THE-ART TECHNIQUES FOR EMBEDDED SOFTWARE ADAPTATION

Suman Roychoudhury, Christian Bunse and Hagen Höpfner
International University in Germany, Campus 3, 76646 Bruchsal, Germany

Keywords: Aspect-oriented Programming, Model-driven Engineering, Embedded Systems.

Abstract: Embedded software systems affect critical functions of our daily lives (e.g., software used in automobiles, aircraft control systems), and represent a significant investment by government, scientific and corporate institutions. Modern research approaches for software engineering and programming language design, such as aspect-oriented software development and model-driven engineering have been investigated as effective means for improving modularization and reuse of software. However, one research trend for embedded system development has focused primarily on the underlying hardware, neglecting the need of applying advanced software engineering techniques for the several million lines of code existing in the embedded domain. In this paper, we evaluate the above mentioned state-of-the-art techniques as a viable solution for the development, analysis and evolution of embedded software systems.

1 INTRODUCTION

Embedded software plays an important role in today's world and is built into electronics in cars, audio equipment, robots, appliances, toys, security systems, televisions and mobile phones. Although there has been a significant amount of research for software modeling, testing, restructuring and transformation-based tools in the non-embedded space, there is a lack of clarity and emphasis for such tools in the embedded domain. The main reason for such isolation is the low-level code base and its greater focus on the underlying hardware (Day, 2005). This has resulted in the use of more informal methods in the embedded space. However with an ever increasing size of embedded software projects, device manufacturers should realize the importance of using modern software engineering tools and techniques that can significantly improve the development, analysis, and maintenance of such software. In this paper, we investigate how aspect-oriented programming (AOP) (Laddad, 2003) and model-driven engineering (MDE) (Schmidt, 2006), can be applied for embedded system development and maintenance.

2 EMBEDDED SOFTWARE ADAPTATION

In this section, we demonstrate how embedded software can be adapted using AOP and MDE techniques. A simple case study application serves as the core example.

The example illustrates a microcontroller (ATMega128) enabled racing car that can be programmed to steer through a predefined track without colliding with obstacles. The car also receives navigational data (i.e., GPS data) during the course of its run. To achieve the desired goal, several hardware devices and sensors were attached to the car as shown in the Figure 1. Two ultrasonic sensors were attached to the front and back of the car for collision avoidance. A GPS device was mounted to provide navigational data (e.g., latitude, longitude, altitude). A photo-interrupter with encoder wheel provided the speed and direction of the car. Finally, a wireless communication link was established between the car and a PC (HyperTerminal) using a Bluetooth device.

In the next sub-section we show how AOP can be applied towards configuring the servos, the sensors and also printing statistics to the HyperTerminal using aspects.

Roychoudhury S., Bunse C. and Höpfner H. (2009).
APPLYING STATE-OF-THE-ART TECHNIQUES FOR EMBEDDED SOFTWARE ADAPTATION .
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 305-308

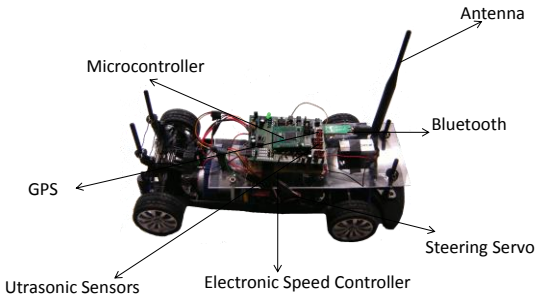

Figure 1: Microcontroller enabled Race Car.

2.1 Adapting AOP Techniques

In the example shown in Figure 1, two servos are used. The electronic speed controller servo is used to control the speed of the car, whereas the steering servo is used to steer the car. The servos are directly connected to the microcontroller and consist of a dc motor, gear train, potentiometer and some control circuitry. To drive the car, pulses are sent continuously to the control circuitry of the servo whose widths vary between 1-2 ms. The potentiometer transforms these pulses into a voltage which represents a certain rotation of the shaft. These values are then compared to average values and the shaft is rotated until both voltages match each other.

Figure 2 illustrates the different control components and the basic interaction functions used to control the servos. The function SetServoPosition is used to send a particular pulse value ranging between 1ms - 2ms to control the speed and direction of the car. In the above scenario, the function SetServoPosition is crosscutting and is being called for every action associated with either the electronic speed servo or steering servo. This crosscutting functionality of the servos can be captured as a single aspect written in AspectC++ (Spinczyk, Gal and Schröder-Preikschat, 2002) as shown in Figure 3.

Note that to steer the car towards centre or left or right, the servo shaft is rotated by 90 degrees or 0 degree or 180 degrees respectively. The corresponding value of the control signal position is passed as `args` to the around advice (see Figure 3).

The advantage of using an aspect allows the servo software to be controlled from one single location. Moreover, this allows finer control of the servo shaft, which can now rotate at any angle between 0-180 degrees instead of predefined positions (left, right or centre) as depicted in Figure 2. Further, along with flexible servo control mechanism, basic statistics could be obtained using

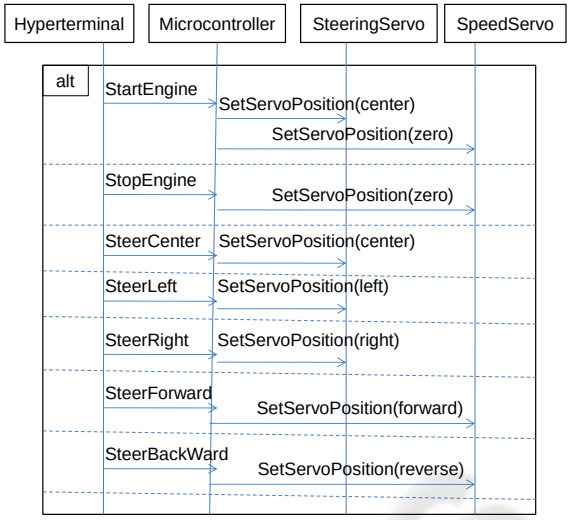

Figure 2: Interaction Diagram for Servo Control.

```
aspect ControlServos {
   unsigned int position;
   pointcut Steer() = void
Steer%(...);
   pointcut StartStopEngine() =
                    void %Engine(...);

   advice execution
        (StartStopEngine ()|| Steer())
        && args(position): around ()
   {
        rprintf("Current Servo
position:

%d",GetServoPosition());
        SetServoPosition (position);
        tjp->proceed();
        rprintf("New Servo position:
                %d", position);
   }
};
```

Figure 3: An Aspect for Capturing Servo Position.

the `rprintf` function as shown in Figure 3. Note that the around advice acts as a wrapper to the existing steering functions.

There are other areas in the car control circuitry where aspects could be useful. It is often the case that several devices share the same clock and require the clock timer to be synchronized. For example, the photo interrupter and encoder wheel, which is connected to measure the car speed and direction, needs to be synchronized with the servo clock timer. A possible solution is to attach a lock-unlock handler as an aspect to the clock timer for all such devices that require synchronization.

2.2 Adapting MDE Techniques

In this section we examine how MDE tools and techniques can ease the construction of embedded software systems. We will investigate two possible directions in that MDE can assist developers in the embedded domain:

- GUI based domain-specific modeling languages
- Text based domain-specific modeling languages

In the first category, we will look into the Embedded System Modeling Language (ESML) (Shi, 2004), which is specifically intended for component-based avionics applications for the Boeing-Boldstroke. In our view, ESML can be customized for other embedded domains as well. ESML allows defining interfaces, events, components, interactions and configurations in the form of UML class diagrams (UML, 2009).

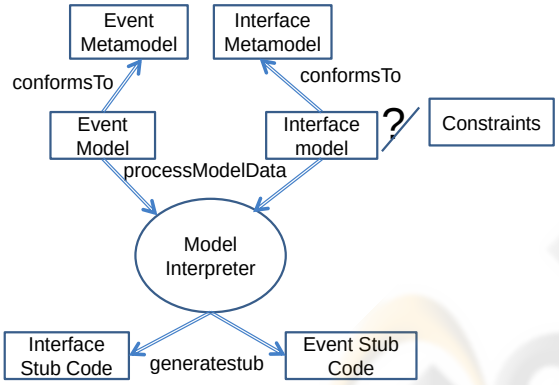

Figure 4: Generative Programming for Embedded Software.

For our case study application, the interface model (see Figure 4) was used to capture various interface methods namely, *GetGPSData*, *GetSonarData*, *GetEncoderData*, and *ReadTimerControl*. The interface model also captures various constraints (e.g., pulses must occur within an interval of 1ms-2ms). Using ESML, event types were used to indicate events such as *StartEngine*, *StopEngine*, *SteerBackward*, *SteerForward*, and *SteerLeft*. Moreover, new events can be added to the existing metamodel in an event repository. The ESML model for our case study application is integrated within the Generic Modeling Environment (Lédeczi et al., 2001). Model interpreters were used to generate the underlying method and event stub code (refer Figure 4). However, the method and event implementation code has to be manually added to the system.

The second dimension using text based domain-specific modeling languages provides another option for embedded system development. Generally, the low-level nature of embedded programs makes them hard to comprehend and difficult to evolve over a period of time. For every hardware or communication channel (e.g., processor type, serial or parallel communication), there is a typical protocol that one needs to follow. Such requirements make the area particularly niche and there is less availability of domain-specific modeling languages (DSMLs) to aid developers to construct embedded software.

We believe that instead of considering the entire embedded domain, sub-domain specific modeling languages can be used, which can only focus the core areas of a sub-domain. For example in the case of our case study application DSMLs can be developed that focuses on the semantics related to the automobile domain only. Similarly for a different sub-domain like mobile systems, a separate DSML model can be used. We carried out our initial investigation with a text-based metamodeling tool suite called ATLAS Model Management Architecture (AMMA).

```
package Automobile {

class RaceCar extends LocatedElement
{
   attribute name: String
   reference sensors[*]
                container: Sensor;
   reference controllers[*]
                container:
Controller;
   reference processors[*]
                container:
Processor;
   reference channels[*]
                container: Channel;
   ...
}
class Sensor extends LocatedElement {
   attribute name : String;
}
class SonarSensor extends Sensor {
   attribute distance : String;
}
class TempSensor extends Sensor {
   attribute temperature : String;
}
...
```

Figure 5: KM3 specification for RaceCar Metamodel.

Using Kernel Meta-metamodel (KM3, part of the AMMA tool suite) (Jouault and Bézivin, 2006), a snippet of the domain-specific metamodel for our case study application is shown in Figure 5. In this figure, the RaceCar class is the main class that defines the abstract syntax of the language. It consists of sensors, controllers, processors, communication channels and other hardware devices. The concrete syntax for the RaceCar model is written in TCS, however, for brevity is not shown here.

Using model transformation techniques, the low-level code required to run the system can be generated for the RaceCar model. Thus, by focusing on graphical or textual based DSMLs, domain engineers can build domain specific embedded tools that can raise the abstraction level of hardware centric embedded programs.

3 SUMMARY AND CONCLUSIONS

The proliferation of embedded software in everyday life has augmented the conformity and invisibility of software. As demands for such software increase, future requirements will necessitate new strategies for improved modularization, construction and restructuring in order to support the requisite adaptations (Masuhara and Kiczales, 2003). Proven software engineering techniques like AOP and MDE that occupy a bigger space in the non-embedded domain must be investigated and disseminated into the embedded space.

In this paper, we demonstrated how each of these techniques can improve the construction effort of embedded software. A greater emphasis must be given to specialized tools and domain specific languages that can raise the abstraction level from hardware centric applications to software centric models and analysis engines.

A growing challenge in the embedded world has been to reduce the energy consumption of battery powered devices. As part of future work, we will look into software centric static and dynamic optimizations embedded mobile systems that combine the power of each of these techniques.

REFERENCES

Day, R. (2005) 'The Challenges of an Embedded Software Enginee', *Embedded Technology Journal*.

Jouault, F. and Bézivin, J. (2006) 'KM3: A DSL for Metamodel Specification', Formal Methods for Open Object-Based Distributed Systems, Bologna, Italy, 171-185.

Laddad, R. (2003) *AspectJ in Action: Practical Aspect-Oriented Programming*, Manning.

Lédeczi, Á., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., Sprinkle, J. and Karsai, G. (2001) 'Composing Domain-Specific Design Environments', *IEEE Computer*, vol. 34, no. 11, November, pp. 44-51.

Masuhara, H. and Kiczales, G. (2003) 'Modeling Crosscutting in Aspect-Oriented Mechanisms', European Conference on Object-Oriented Programming, Springer-Verlag LNCS 2743, Darmstadt, Germany, 2-28.

Schmidt, D. (2006) 'Model-Driven Engineering', *IEEE Computer*, February.

Shi, F. (2004) *Embedded Systems Modeling Language*,http://www.omg.org/news/meetings/workshops/MIC_2004_Manual/06-4_Shi_etal.pdf.

Spinczyk, O., Gal, A. and Schröder-Preikschat, W. (2002) 'AspectC++: An Aspect-Oriented Extension to C++', International Conference on Technology of Object-Oriented Languages and Systems, Sydney, Australia, 53-60.

UML (2009) *Unified Modeling Language 2.2*, http://www.omg.org/technology/documents/formal/uml.htm.